МОДЕЛІ ЕЛЕКТРОННОГО НАВЧАННЯ ТА ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

УДК 004.89:519.8

ГОРБАЧ Тетяна Вікторівна
науковий співробітник кафедри програмної інженерії ХНУРЕ
Наукові інтереси: Розробка і впровадження комп'ютерних навчальних програм, розробка алгоритмічного та програмного забезпечення для дистанційної освіти, розробка програмних засобів мультимедіа.
e-mail: yova.tanya@gmail.com.

СЛАВГОРОДСЬКИЙ Владислав Юрійович
аспірант кафедри програмної інженерії Харківського національного університету радіоелектроніки
Наукові інтереси: Математичне моделювання, розробка і впровадження комп'ютерних навчальних програм, розробка алгоритмічного та програмного забезпечення для дистанційної освіти
e-mail: kearir12@gmail.com.

ШУБІН Ігор Юрійович,
кандидат технічних наук, професор кафедри програмної інженерії Харківського національного університету радіоелектроніки
Наукові інтереси: Розробка і впровадження комп'ютерних навчальних програм, розробка алгоритмічного та програмного забезпечення для дистанційної освіти, розробка програмних засобів мультимедіа.
e-mail: igor.shubin@nure.ua.

КОВАЛЕВСЬКА Алла Володимирівна
доцент кафедри економіки підприємств міського господарства Харківського національного університету міського господарства імені А.Н. Бекетова
Наукові інтереси: Математичне моделювання, розробка алгоритмічного та програмного забезпечення для дистанційної освіти, розробка програмних засобів мультимедіа, економіко-математичне моделювання методів вирішення управлінських завдань.
e-mail: torpal@ukr.net.

ВСТУП

Сучасний стан освітньої системи України характеризується активними змінами на шляху до євроінтеграції та міжнародної співпраці в області вищої освіти. З цього погляду активізація наукових досліджень з питань ефективного управління на рівні вищого навчального закладу, зокрема на основі використання прогресивних інформаційних технологій, є важливим завданням сьогодення. Розвиток новітніх інформаційних технологій в освіті неодмінно спричиняє появу нових форм навчання. Поряд із очною та заочною формою дедалі частіше використовують електронну освіту. Відсутність стандартів та уніфікованого підходу до створення систем електронного навчання в Україні впливає не лише на ринок таких програмних продуктів, але й на якість дистанційної освіти. Одним із шляхів вирішення проблеми якості навчання є створення окремих педагогічних програмних засобів для кожного предмету. Але цей процес є дуже витратним як з огляду на матеріальні, так і на людські ресурси. Отже, постає питання розробки програмного забезпечення, яке б задовольняло вимоги щодо організації освітнього процесу на основі методології електронного навчання.

Огляд процесу розробки програмного забезпечення. Як правило, всі системи дистанційного навчання мають модульну структуру. Навіть якщо фізич-

но модулі важко виділити, вони легко виокремлюються на концептуальному рівні та рівні розробки. До таких модулів належить навчально-методичний модуль, який реалізує розроблення і публікацію навчального матеріалу; педагогічний модуль, який реалізує функції структурування і послідовності подання навчального матеріалу (НМ) та моделювання зв'язку студента з підсистемою контролю знань; інтерфейсний модуль (навігація між елементами навчального матеріалу); комунікаційний модуль (функції інтерактивної взаємодії та зворотного зв'язку); організаційно-адміністративний модуль (функції авторизації, захисту, фінансових операцій та інші).

Визначені покоління електронного та дистанційного навчання розрізняються не тільки наявністю того або іншого модуля, але й їхнім внутрішнім змістом та сукупністю зв'язків між ними. Знання предметної області, сформовані навчально-методичним модулем, організовані у відповідні інформаційні одиниці навчального матеріалу, визначають предмет вивчення. Педагогічний модуль визначає оптимальну послідовність подання навчального матеріалу. Крім того, до складу педагогічного модуля входить підсистема контролю знань студента, зазвичай у вигляді тестів [1]. У системах дистанційного навчання нового покоління рівень знань студента оцінюється також безпосередньо під час вивчення навчальних матеріалів. Модель підвищує ефективність навчання, оскільки знаючи кого навчати, навчальний процес максимально індивідуалізується, тобто адаптується. Інтерфейсний модуль забезпечує зв'язування інших модулів системи дистанційної освіти, а комунікаційний забезпечує взаємодію та зворотний зв'язок студента із системою. Організаційно-адміністративний модуль (або підсистема керування навчальним процесом) забезпечує функції авторизації, захисту, здійснення фінансових операцій та інші.

Еталонна модель об'єкта контенту для спільного використання (SCRM). Sharable Content Object Reference Model – еталонна модель об'єкта контенту для спільного використання – створення цього стандарту є першим кроком на шляху розвитку концепції Advanced Distributed Learning (ADL), оскільки цей стандарт визначає структуру навчальних матеріалів і інтерфейс середовища виконання, за рахунок чого навчальні об'єкти можуть бути використані в різних системах дистанційного і комп'ютерного навчання.

Основними цілями проекту є надання освітньому співтовариству таких можливостей:

1) доступність – можливість пошуку і доступу до об'єктів, що знаходяться в різних місцях;

2) адаптивність – можливість налаштовувати навчання під індивідуальні або організаційні потреби;

3) ефективність – можливість скоротити час і вартість доставки знань до студентів;

4) здатність до перенесення – можливість переносити об'єкти, створені одним набором засобів розробки або платформою, на інші і використовувати без змін;

5) захист інвестицій – зміни технологій не ведуть до необхідності переробки об'єктів;

6) повторне використання об'єктів – можливість складати курси з найдрібніших об'єктів, гнучкість при використанні в різних контекстах.

Всі ці принципи успішно можуть бути дотримані в тому випадку, якщо з самого початку орієнтуватися на використання освітнього контенту в веб-середовищі [2].

Оскільки Web є ідеальним середовищем поширення та використання освітніх матеріалів, тому розробники SCORM зробили його сумісним з можливостями мережі з таких причин:

1) web-технології та інфраструктура швидко розширюють можливості освітніх технологій;

2) web-стандарти освітніх технологій ще не існують в розповсюдженій формі;

3) web-контент можна поширювати і використовувати в будь-якому середовищі (автономні системи або мережеві середовища).

ОПИС СИСТЕМИ

Система подається як набір сервісів, які конфігуруються залежно від вимог замовника. Замовником може виступати фізична або юридична особа, якій потрібно автоматизувати процеси навчання. Система будується з автономних сервісів, які є незалежними один від одного. Спілкування між сервісами відбувається за допомогою HTTPS протоколу. Також замовник може надати свій сервіс, який підтримує контракт системи. Сервіси є незалежними та можуть бути реалізовані різними мовами програмування та розміщені на

різних апаратних платформах. Для повноцінного функціонування продукту програмне забезпечення розробляється з можливістю підключення до п'ятдесяти модулів одночасно, а також його надійності, відкритості до змін, захищеності.

В контексті роботи під системою розуміється проста взаємопов'язана структура, яка складається із таких компонентів:

1) сервіс зв'язку компонентів;
2) навчально-методичний сервіс;
3) педагогічний сервіс;
4) сервіс контролю знань;
5) сервіс авторизації та аутентифікації;
6) сервіс оповіщень.

Структура системи представлена на рис. 1.

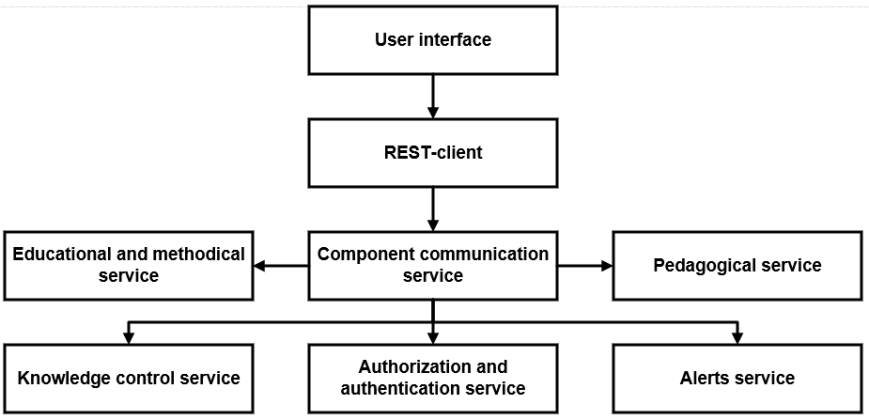

Рисунок 1 – Структура системи

Оскільки сервіси є змінними, то надалі будемо розглядати лише головний (незмінний) сервіс, а саме сервіс зв'язку компонентів.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги описують внутрішню роботу системи, її поведінку (калькулювання даних, маніпулювання даними, опрацьовування даних) та інші специфічні функції, які повинна виконувати система. Функціональні вимоги визначають, що система повинна робити [3].

Сервіс є головним вузлом, який приймає запити з REST-клієнта та перенаправляє на потрібний сервіс. Також він виступає внутрішнім маршрутизатором, який забезпечує комунікацію між внутрішніми сервісами. Для того щоб сервіс мав інформацію про сервіси всередині системи та їх контракти, при старті додатку відбувається реєстрація всіх сервісів.

До функціональних вимог сервісу зв'язку компонентів (рис. 2) належать:

1) створити конфігураційний файл;
2) редагувати конфігураційний файл;
3) видалити конфігураційний файл;
4) зареєструвати сервіси;
5) ввімкнути сервіс;
6) вимкнути сервіс.

Вимога «Створити конфігураційний файл» надає можливість створення файлу за конфігурацією. Для цього слід вказати назву файлу та додати основні атрибути, такі, як хости, порти та назви сервісів, які будуть використовуватися при такій конфігурації.

Прецедент «Редагувати конфігураційний файл» включає в себе вибір файлу, який необхідно змінити.

Прецедент «Видалити конфігураційний файл» включає в себе вибір файлу, який є неактуальним та потребує видалення.

Варіант використання «Зареєструвати сервіси» дозволяє зареєструвати всі сервіси, між якими потрібно налаштувати комунікацію, для цього слід вказати конфігураційний файл з описом доступних сервісів.

«Ввімкнути сервіс» – прецедент, що дозволяє ввімкнути сервіс та зробити його видимим для інших. Слід зазначити, що сервіс має бути зареєстрованим.

«Ввімкнути сервіс» – прецедент, що дозволяє вимкнути сервіс та зробити його недоступним для інших, при цьому сервіс має бути зареєстрованим.

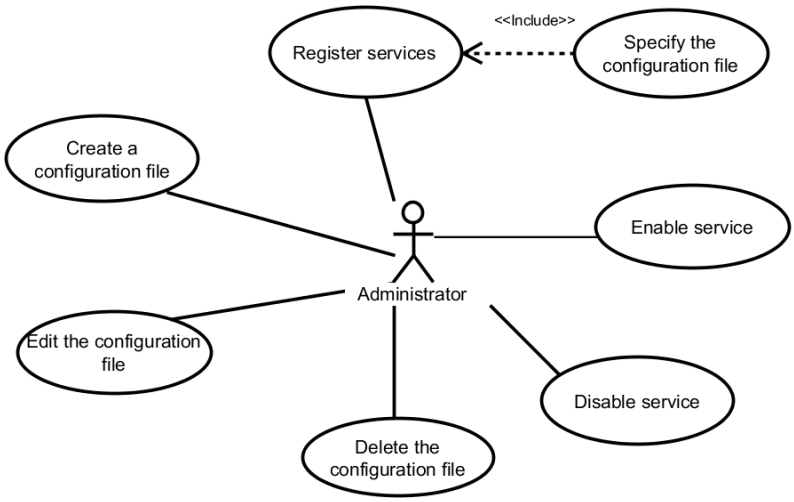

Рисунок 2 – Діаграма варіантів використання для сервісу зв'язку компонентів

Нефункціональні вимоги можна поділити на дві категорії: покращення (безпека, надійність, швидкодія та інше) та вдосконалення (масштабування, відновлюваність та ін.) властивостей системи [4].

Було виділено найбільш важливі нефункціональні вимоги.

Надійність – це властивість програмної системи функціонувати з рівнем помилок (збоїв), що не перевищує заданого рівня.

Основні вимоги до надійності сервісу:

1) при відмові одного із сервісів, сервіс зв'язку повинен дати коректну відповідь клієнту;

2) перенаправлення запиту на потрібний сервіс не повинен займати більше 1 секунди.

Здатність до перенесення – вимога для програмного додатка або системи щодо переносу на інші платформи. Головною вимогою до портативності сервісу є підтримка незалежності від операційної системи та робота із такими ж незалежними сервісами.

Вимоги до безпеки:

1) обмін даними між сервісами та клієнтом повинен відбуватися у вигляді передачі зашифрованих даних із ключем не меншим ніж 128 біт за технології Secure Socket Layer (SSL) через захищене HTTP з'єднання (HTTP8);

2) операційна система на сервісах повинна забезпечити ідентифікацію та аутентифікацію адміністратора операційної системи при його локальних запитах на доступ;

3) всі паролі повинні передаватися в зашифрованому вигляді.

Масштабованість – можливість збільшувати продуктивність системи пропорційно до виділених ресурсів, буває горизонтальною і вертикальною. Горизонтальне масштабування – це збільшення кількості серверів, що взаємодіють один з одним в прозорому режимі для розподілу загального завантаження системи. Вертикальне масштабування досягається за рахунок збільшення потужності окремого сервера або заміною апаратного забезпечення на більш швидкодіюче. Таким чином, система (сервіс) має можливість горизонтального та вертикального масштабування.

Моделювання послідовності виконання. Діаграма послідовності відображає взаємодії об'єктів, впорядкованих у часі. Це діаграма, на якій для деякого набору об'єктів на єдиній тимчасовій осі показані життєвий цикл і взаємодія модулів системи.

Основними елементами діаграми послідовності є позначення об'єктів, вертикальних «ліній життя», що відображають плин часу, прямокутників, що відображають діяльність об'єкта або виконання ним певної функції і стрілки, що показують обмін сигналами або повідомленнями між об'єктами. На рисунку 3 зображено діаграму послідовності для проходження тесту.

На діаграмі можна спостерігати взаємодію користувача з браузером, а також браузера з сервісом комунікацій, який вирішує куди перенаправити запит, та іншими сервісами. В сервісі комунікації відбувається

перевірка доступності сервісу для обробки запиту та перенаправлення запиту [5].

Вибір архітектури програмного забезпечення. Зважаючи на вимоги, які були описані вище, розглянемо такі підходи до розробки архітектури, як еталонна трирівнева архітектура та мікросервісна архітектура.

Трирівнева архітектура – архітектурна модель програмного комплексу, що допускає наявність в ньому трьох компонентів: клієнта, сервера додатків (до якого підключено клієнтський додаток) і сервера баз даних (з яким працює сервер додатків).

Клієнт – це інтерфейсний компонент комплексу, що надається кінцевому користувачеві. Цей рівень не повинен мати прямий зв'язок з базою даних, містити основну бізнес-логіку і зберігати стан програми. На цей рівень зазвичай виноситься тільки найпростіша бізнес-логіка: інтерфейс авторизації, алгоритми шифрування, перевірка вхідних даних на допустимість і відповідність формату, нескладні операції з даними.

Сервер додатків розташовується на другому рівні, в ньому зосереджена велика частина бізнес-логіки. Поза ним залишаються тільки фрагменти, експортовані на клієнта, а також елементи логіки, поміщені до бази даних. Сервери додатків проектуються таким чином, щоб додавання до них додаткових екземплярів забезпечувало горизонтальне масштабування продуктивності програмного комплексу і не вимагало внесення змін в програмний код додатка.

Сервер баз даних забезпечує зберігання даних і виноситься на окремий рівень; реалізується, як правило, засобами систем управління базами даних, підключення до цього компоненту забезпечується тільки з рівня сервера додатків [4].

На рисунку 4 показано еталонну трирівневу систему архітектури «клієнт-сервер» з виділеним сервером додатків.

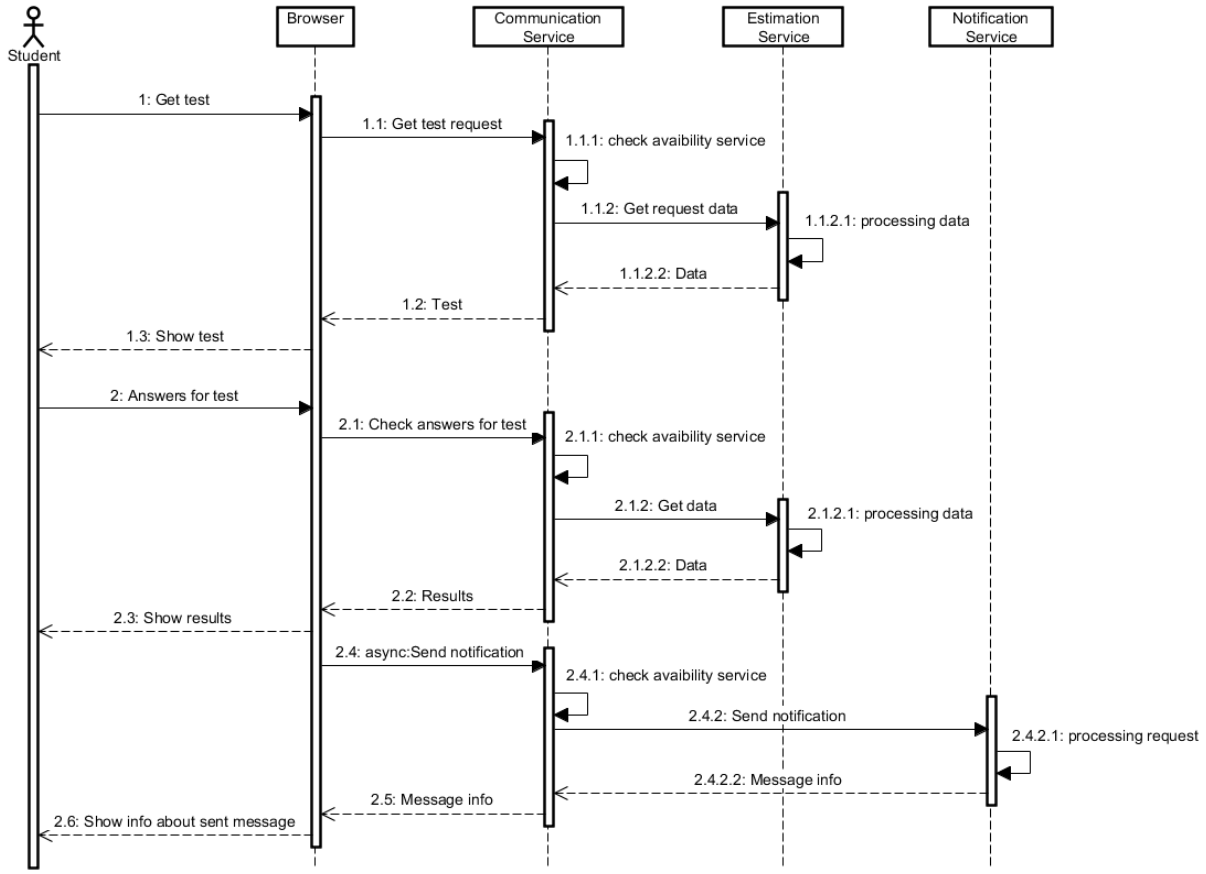

Рисунок 3 – Діаграма послідовності

Вузол Client має компонент «WebBrowser» – це програма, яка дозволяє користувачу отримати доступ до інтернету та набір бібліотек JavaScript, які полегшують роботу на стороні «Client».

Вузол WebServer – це веб-сервер системи, клієнт взаємодіє з ним за допомогою протоколу HTTPS. Розмірність відповідного зв'язку вказано як «0 .. *» до «1», тобто, передбачена можливість одночасної роботи декількох клієнтів з одним web-сервером.

Вузол DataBase Server – це вузол розміщення бази даних системи, яка реалізована за допомогою конкретної MySQL, зв'язок з цим вузлом забезпечується за допомогою JDBC драйвера.

Мікросервісний стиль архітектури – підхід до розробки цілісної програми як набору маленьких сервісів, кожен із яких працює у власному процесі та з'єднується з іншими за допомогою легких механізмів, таких, як HTTP. Сервіси будуються відповідно до певної задачі та можуть незалежно розгортатись автоматизованими системами.

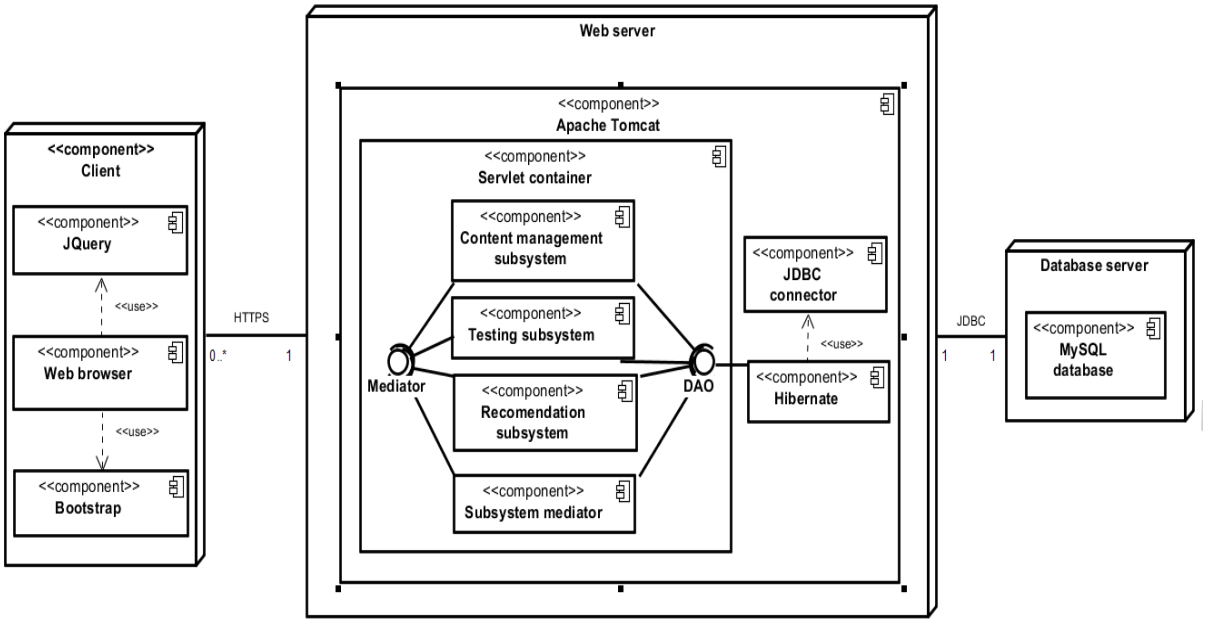

Рисунок 4 – Еталонна трирівнева архітектура системи

Уся логіка обробки запитів працює в одному процесі, що дозволяє використовувати наявні інструменти мови програмування для поділу програми на класи, функції та простори імен.

Мікросервіси можна розгортати та масштабувати незалежно один від одного. Вони мають чіткі межі між модулями та дозволяють реалізовувати окремі підсистеми різними мовами програмування. Таке розмежування допомагає управляти складністю, оскільки кожен модуль матиме публічний API, який містить лише потрібну функціональність, а все інше буде інкапсульовано та не мати значення для розробки інших сервісів, які залежать від цього [6].

На рисунку 5 зображена діаграма компонентів для системи при мікросервісному підході.

Як показано на діаграмі, всі сервіси є незалежними один від одного та взаємодіють лише на основі відкритого інтерфейсу. Також на діаграмі зображений API GateWay, який виступає своєрідним проксі сервером. Це надає зручності роботи з системою, оскільки звернення відбувається за однією адресою, а не за індивідуальною адресою кожного мікросервісу. Також використання API GateWay інкапсулює мікросервіси, що в свою чергу дозволяє змінювати їх структуру та API.

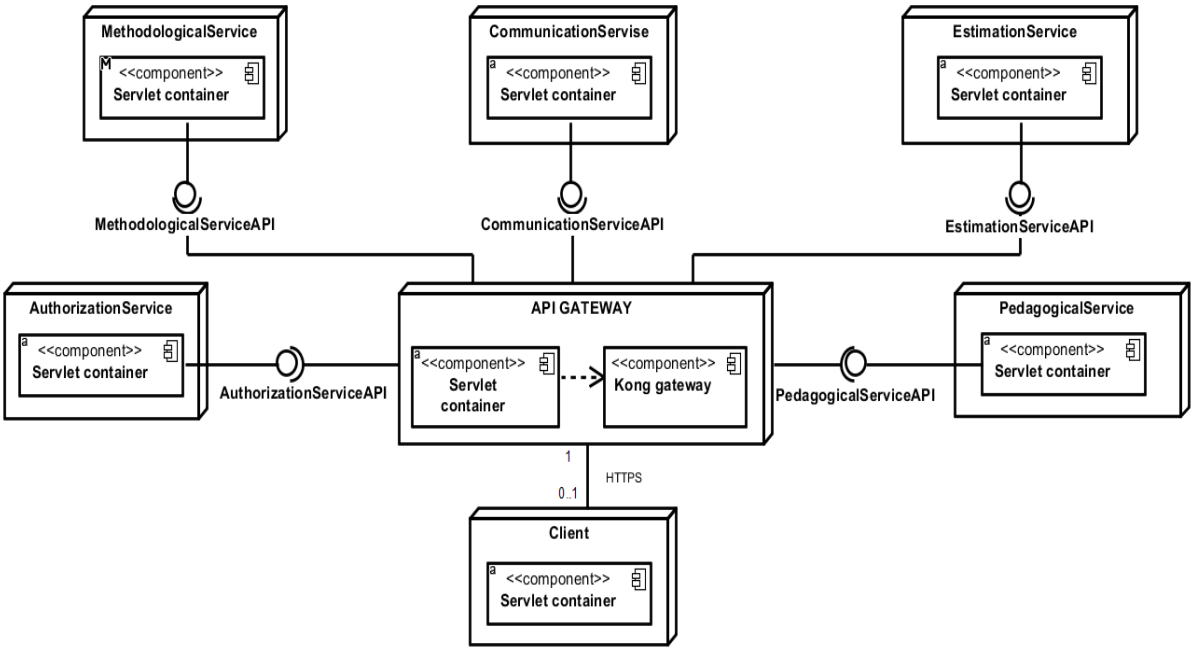

Рисунок 5 – Діаграма компонентів при мікросервісному підході

Еталонні рішення. Функціональна декомпозиція додатка є ключем до створення успішної мікросервісної архітектури. Це дозволяє досягти слабкої зв'язності та високої згуртованості (декілька менших сервісів можуть створювати один більший композитний, вищий за рівнем сервіс). Функціональна декомпозиція дає швидкість, гнучкість, масштабованість та інші переваги, але її метою є створення архітектурного рішення, яке було б стійким до змін.

Еталонним рішенням або патерном проектування називають рішення для проблем, які часто зустрічаються в області розробки програмного забезпечення. Патерни проектування не є готовими рішеннями, які можна трансформувати безпосередньо в код, а являють собою загальний опис вирішення проблеми, який можна використовувати в різних ситуаціях.

За класифікацією GoF патерни розрізняють за типами і видамb. Існує кілька типів патернів проектування, кожен з яких призначений для вирішення свого кола завдань [7]:

1) породжувальні патерни, призначені для створення нових об'єктів в системі;

2) структурні патерни, які вирішують задачі компонування системи на основі класів і об'єктів;

3) патерни поведінки, призначені для розподілу обов'язків між об'єктами в системі.

Агрегатор є найбільш поширеним патерном при проектуванні мікросервісів. У найпростішій формі агрегатор буде звичайною веб-сторінкою, яка викликає декілька служб для досягнення функціональності, якої вимагає додаток. Кожен із мікросервісів надає легковісний REST, за допомогою якого веб-сторінка може отримати дані та обробити або відобразити їх відповідним чином (рис. 6).

Інший варіант шаблону агрегатора – коли не треба відображати дані, а замість цього потрібно просто створити композитний мікросервіс, який буде використовуватись іншими сервісами. В цьому випадку агрегатор повинен зібрати дані від інших мікросервісів, застосувати бізнес-логіку до них та в подальшому опублікувати їх як кінцеву точку REST. Ці дані потім можуть бути використані іншими сервісами, які їх потребують [8].

Посередник – поведінковий шаблон проектування, що забезпечує взаємодію множини об'єктів, формуючи при цьому слабку зв'язаність і позбавляючи об'єкти необхідності явно посилатися один на одного.

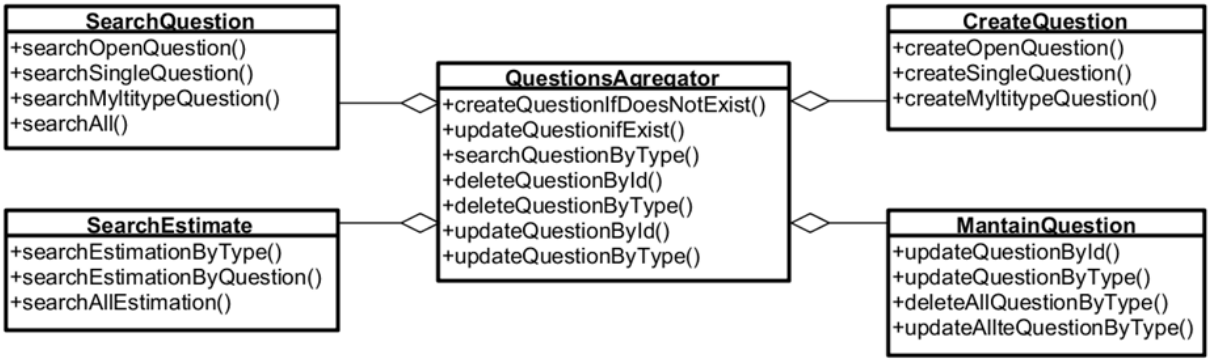

Рисунок 6 – Використання агрегатора мікросервісів

Оскільки система, що проектується, складається з трьох підсистем (підсистеми управління контентами, підсистеми тестування, підсистеми аналізу результатів та рекомендацій), виникає необхідність організації взаємодії між ними [8].

Патерн-посередник використовується для того, щоб позбутися необхідності явної взаємодії підсистем. Натомість функцію взаємодії бере на себе посередник (рис. 7).

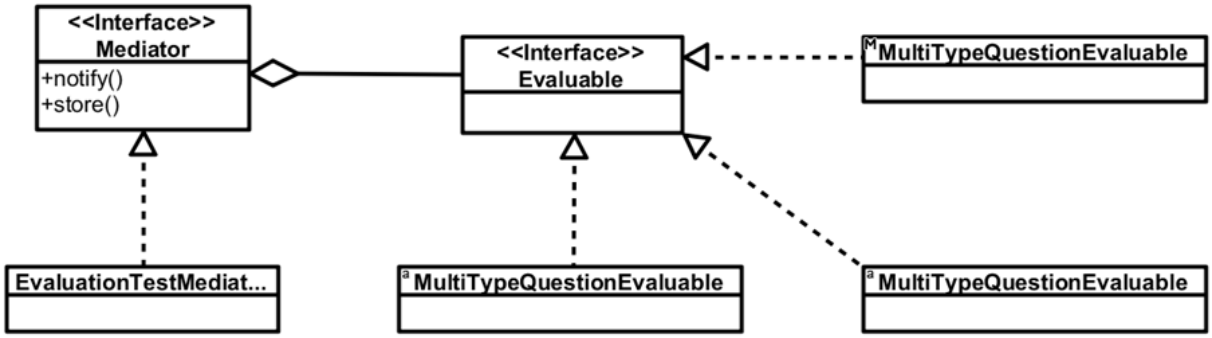

Рисунок 7 – Використання посередника для взаємодії між підсистемами

Превагами застосування цього патерну є зменшення числа породжуваних підкласів, оскільки посередник інкапсулює поведінку, яку необхідно було б розподіляти між декількома об'єктами. Для зміни поведінки необхідно лише породжувати посередника, а «колег» можна використовувати повторно [7, 8].

Окрім того, до переваг даного рішення відноситься усунення зв'язку між колегами. Це дає можливість змінювати класи колег та посередника незалежно один від одного.

Мінусом даного підходу є централізація управління. Посередник переносить всю складність взаємодії в себе, що робить його монолітним та складним для розуміння [7].

Ланцюговий шаблон проектування мікросервісів надає єдину відповідь на певний запит. У цьому випадку запит від клієнта отримано сервісом A, який в свою чергу з'єднаний із сервісом B, який може мати зв'язок з сервісом C. Всі сервіси, ймовірніше всього, використовують синхронний обмін повідомленнями по HTTPS.

Слід враховувати те, що клієнт буде заблокований, поки не закінчиться повний ланцюжок запитів / відповідей між сервісами. Класи, які є складовою ланцюжка, містять у собі метод обробки даних, а також поле, в якому міститься посилання на наступного обробника в ланцюжку [7].

Асинхронний обмін повідомленнями. Водночас, коли використання REST є дуже поширеним та добре зрозуміли, воно накладає обмеження на те, щоб бути синхронним та неблокуючим. Асинхронність може бути досягнена, але тільки специфічним прикладним шляхом. У деяких випадках мікросервісні архітектури мо-

жуть використовувати черги повідомлень замість запитів / відповідей у REST.

У цьому шаблоні проектування сервіс А може викликати сервіс С синхронно, який в свою чергу зв'язаний з сервісами В і Б в асинхронному режимі з використанням загальної черги повідомлень. Зв'язок А та С може бути асинхронним, можливо, з використанням WebSocket, для досягнення бажаної масштабованості. Поєднання запитів/відповідей у REST із чергами повідомлень може бути використано для реалізації задач бізнес-логіки [7].

ВИЗНАЧЕННЯ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ТА ФУНКЦІОНУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Аналізуючи наведені вище діаграми та специфіки програмного рішення, яке моделюється, було обрано мову програмування та платформу виконання Java, а також такий стек технологій:

1. JDK 1.8 – комплект розробника додатків мовою Java, який включає в себе компілятор Java, стандартні бібліотеки класів Java, приклади, документацію, різноманітні утиліти і виконавчу систему Java (JRE). До складу JDK не входить інтегроване середовище розробки на Java, тому розробник, що використовує тільки JDK, повинен використовувати текстовий редактор і компілювати та виконувати свої програми через утиліти командного рядка.

2. Intellij Idea 15.0 Community Edition – інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та ін.) від компанії JetBrains. Система поставляється у вигляді урізаної за функціоналом, безкоштовної версії «Community Edition» і повнофункціональної комерційної версії «Ultimate Edition», для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. Сирцеві тексти Community-версії поширюються рамках ліцензії Apache 2.0 [9].

3. Spring Framework – програмний каркас (фреймворк) з відкритим кодом та контейнером з підтримкою інверсії управління для платформи Java. Основні особливості Spring Framework можуть бути використані будь-яким застосунком Java, але є розширення для створення веб-додатків на платформі Java Enterprise Edition не нав'язує будь-яку конкретну модель програмування.

4. Spring Cloud Config – це сховище конфігурацій, що горизонтально масштабується для розподіленої системи. Як джерело даних на даний момент підтримуються Git, Subversion і прості файли, що зберігаються локально. За замовчуванням Spring Cloud Config віддасть файли, відповідного імені запитувача Spring-додатка (але можна забирати властивості залежно від конкретного Spring-профілю і з певно] гілки системи контролю версій).

5. Kong – масштабований API шлюз. Kong проходить перед будь-яким RESTful API і поширюється через плагін, який забезпечує додаткові функціональні можливості і послуги за межами основної платформи.

6. jQuery – JavaScript-бібліотека з відкритим вихідним кодом.

Для розробки програмного забезпечення на Java Enterprise Edition краще за все використовувати Intellij Idea від NetBeans.

Intellij IDEA – комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala та ін.) від компанії NetBeans. Система поставляється у вигляді урізаної за функціональністю, безкоштовної версії «Community Edition» і повнофункціональної комерційної версії «Ultimate Edition», для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. Вихідний код Community-версії поширюються в рамках ліцензії Apache 2.0. Бінарні складові підготовлені для Linux, Mac OS і Windows.

Основними можливостями розробки Community-версії середовища Intellij IDEA, є підтримка інструментів для проведення тестування TestNG і JUnit, системи контролю версій Subversion, Mercurial і Git, засобів складання Maven і Ant, мов програмування Java, Scala, Clojure, Groovy і Dart. Підтримується розробка додатків для мобільної платформи Android.

Є доступними засоби інтеграції з системами відстеження помилок JIRA, Trac, Redmine, Pivotal, Tracker, GitHub, YouTrack, LightHouse.

Тестування програмного забезпечення – це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має виконуватись. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки, при цьому може оцінюватись:

1) відповідність вимогам, якими керувалися проектувальники та розробники;

2) правильна відповідь для усіх можливих вхідних даних;

3) виконання функцій за прийнятний час;

4) практичність;

5) сумісність з програмним забезпеченням та операційними системами;

6) відповідність задачам замовника [8].

Оскільки число можливих тестів навіть для нескладних програмних компонент практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат, ПЗ тестується стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість ПЗ, ризики відмови як для користувачів, так і для замовників.

Тестування може проводитись, як тільки створено код. Процес розробки зазвичай передбачає час для тестування. Наприклад, при поетапному процесі більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно. Як висновок, можна зауважити, що тестування ПЗ – техніка контролю якості, що перевіряє відповідність між реальною і очікуваною поведінкою програми завдяки кінцевому набору тестів, які обираються певним чином.

Перш за все, тестування поділяється на дві категорії: автоматичне і ручне.

У цій роботі використовувались обидва види тестування, однак ручне тестування не було задокументовано. Як автоматичне було використано модульне та інтеграційне тестування.

Модульне тестування або **юніт-тестування** – процес в програмуванні, який дозволяє перевірити коректність певних модулів усієї програми. Ідея полягає в тому, щоб писати тести для кожної нетривіальної функції чи методу. Це дозволяє достатньо швидко перевірити, чи не привели певні зміни до регресії, тобто до появи помилок у вже протестованих місцях програмного рішення, а також прискорює пошук та усунення таких помилок. Таким чином, юніт-тестування являє собою перший бастіон у боротьбі за якість програмного забезпечення [9, 10].

Модуль в контексті модульного тестування являє собою мінімальну смислову одиницю вихідного коду, придатну для тестування (функція, процедура, метод, в окремих випадках – клас). Модульний тест перевіряє працездатність модуля в умовах ізоляції від інших модулів програми. Виключивши вплив інших модулів, модуль перевіряється в стерильних умовах, якщо результат виконання модульного тесту не відповідає очікуваному, то помилка криється в логіці модуля, а не в зовнішніх чинниках. Приклад таких тестів для програмного додатку зображений на рис. 8.

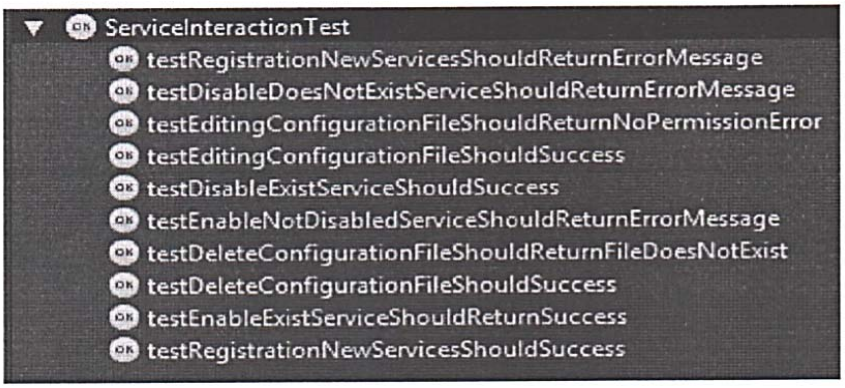

Рисунок 8 – Приклад модульних тестів

Інтеграційне тестування – це фаза тестування ПЗ, під час якої окремі модулі системи комбінуються та тестуються разом, у взаємодії. Інтеграційне тестування виконується після модульного тестування та перед

верифікацією та валідацією ПЗ. Якщо розглядати цей процес як систему, то на вхід їй подаються модулі, які вже пройшли модульне тестування, потім модулі групуються в більші частини, виконуються тести передбачені планом, а на виході системи – інтегрована система, що готова до системного тестування або тестування сценаріїв.

Метою інтеграційного тестування є верифікація вимог з функціональності, продуктивності, надійності до основних компонентів програми. Ці компоненти, тобто групи модулів, тестуються методом «чорної скриньки», успішні та неуспішні тест-кейси симулюються відповідними вхідними параметрами. В процесі інтеграційного тестування тестуються симульоване використання спільних даних та комунікація між процесами. Тест-кейси перевіряють, чи коректно взаємодіють всі компоненти, наприклад, через виклик процедури чи активізацію процесу. Дуже важливо те, що до інтеграційного тестування приступають тільки після модульного. Це дозволяє реалізувати стратегію «будівельних блоків», коли до верифікованої системи інтегруються верифіковані модулі.

В цілому, інтеграційне тестування перевіряє інтерфейси між компонентами, взаємодію з різними частинами системи, такими, як операційна система, файлова система та апаратне забезпечення, інтерфейси між системами. Інтеграційне тестування може складатися з одного чи більше рівнів та може виконуватися на тестових об'єктах різного розміру.

Приклади прецедентів (Test cases) для інтеграційного тестування подані в додатку.

Приклад автоматичних інтеграційних тестів відносно розроблюваного веб-додатка зображений на рис. 9.

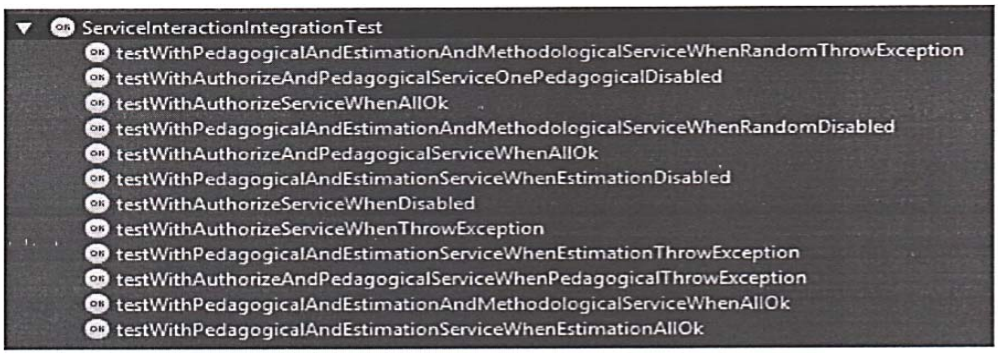

Рисунок 9 – Приклад автоматичних інтеграційних тестів

В ході інтеграційного тестування всі тести пройшли вдало. Помилок у взаємодії підсистем не виявлено.

Опис вказівок для користувача. Розроблюване програмне рішення створене для підприємств, які хочуть поліпшити знання свої працівників. Це програмне забезпечення являє собою веб-додаток. Для того щоб запустити веб-додаток на персональному комп'ютері, потрібно мати набір засобів для розробника, таких, як Apache TomCat контейнер сервлетів та СКБД MySQL, але якщо додаток буде розміщений в інтернеті, то для того, щоб почати використовувати його, потрібно просто перейти за посиланням.

Перш за все, для аутентифікації користувача необхідно зареєструватись у системі. Форма для заповнення зображена на рис. 10.

Рисунок 10 – Форма реєстрації

Для того щоб зареєструватися, користувач має ввести валідний логін, e-mail, пароль, вказати роль в системі, а також ввести капчу.

Після успішної реєстрації можна ввійти в систему. На рис. 11 зображена форма авторизації.

Для авторизації необхідно ввести логін та пароль, а також капчу. Користувач може обрати опції «Remember me».

Після успішної авторизації користувача буде перенаправлено до дошки приладів. Розглянемо можливості системи під різними ролями.

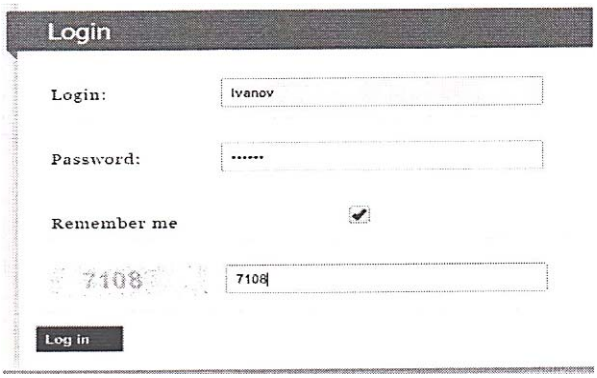

Рисунок 11 – Форма авторизації

Спочатку розглянемо можливості системі для ролі «Викладача». На рис. 12 зображено меню користувача.

Головними можливостями в системі є створення курсу та редагування курсів, перегляд студентів, а також комунікація з ними.

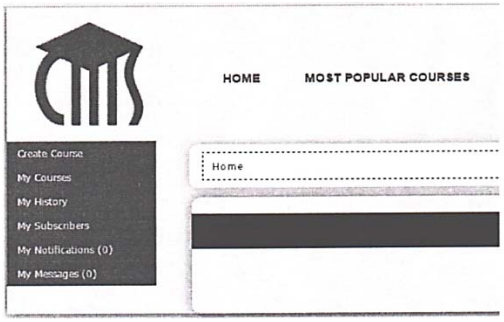

Рисунок 12 – Меню користувача

На рисунку 13 зображена форма створення курсу.

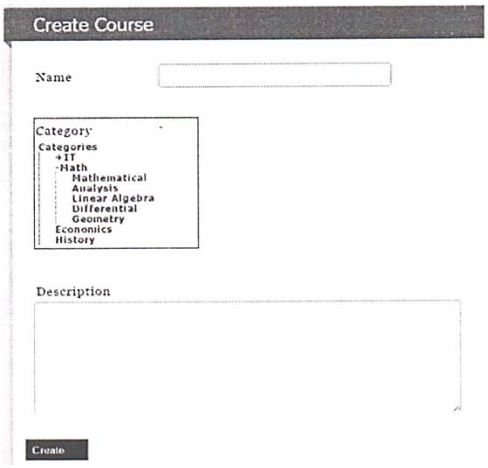

Рисунок 13 – Форма створення курсу

Для того щоб створити курс, необхідно вказати його назву, вибрати до якої категорії він належить, а також задати його опис. Після цього він з'явиться в меню з курсами та буде доступний для редагування (рис. 14).

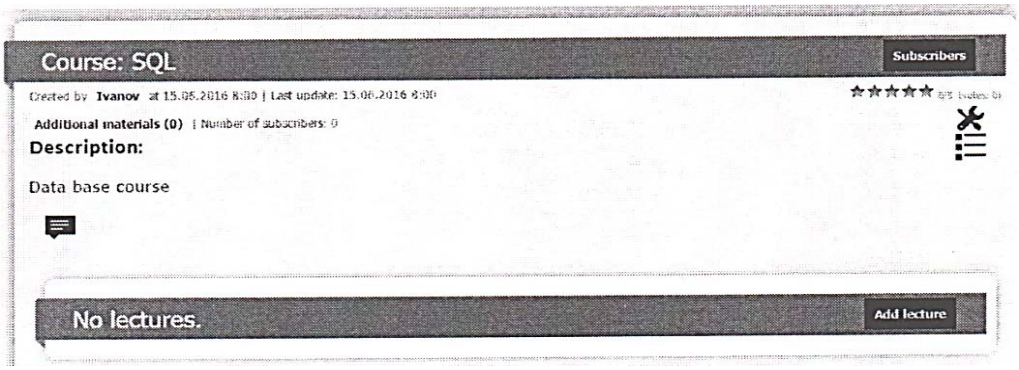

Рисунок 14 – Меню редагування курсу

До основних можливостей роботи з курсом належать перегляд студентів, які його проходять, додавання лекції, а також редагування інформації про курс.

На рисунку 15 зображено створення нової лекції до курсу.

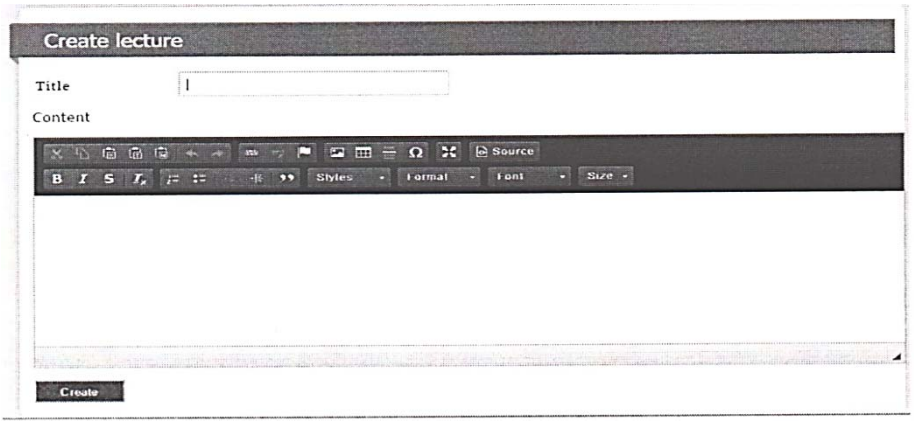

Рисунок 15 – Створення нової лекції

Після створення лекцію можна знайти в меню курсу. Для кожної лекції можна задати свій набір тестів (рис. 16).

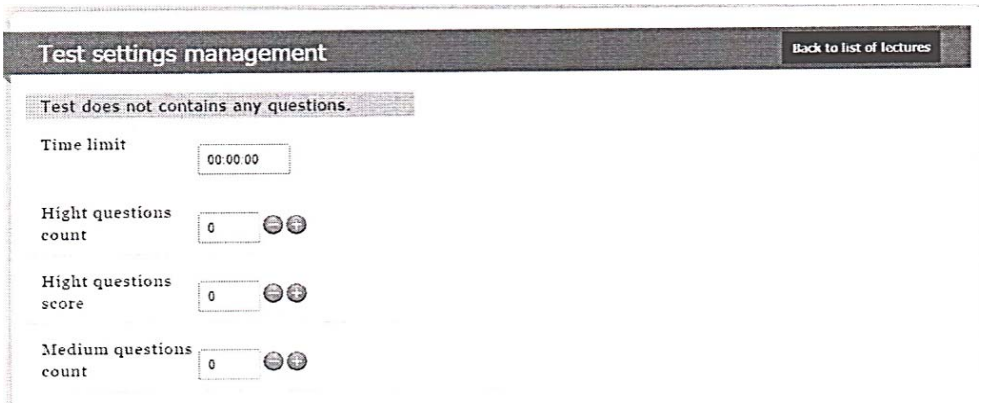

Рисунок 16 – Створення нового тесту

Далі розглянемо можливості системи з роллю «Студент». На дошці приладів студента відображається головне меню, а також останні додані курси (рис. 17).

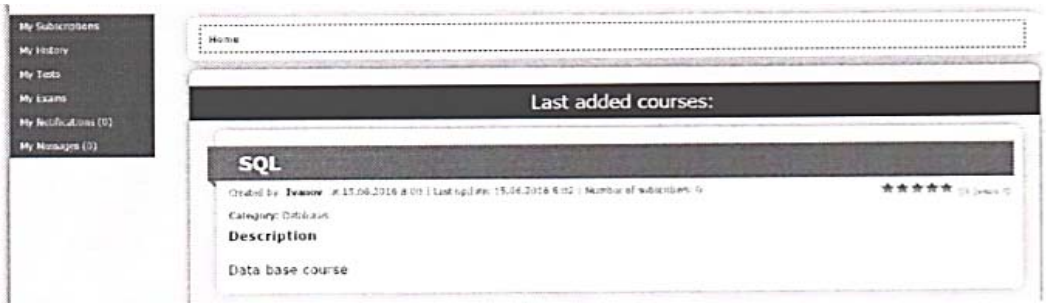

Рисунок 17 – Дошка приладів «Студента»

Головними можливостями системи у ролі «Студент» є перегляд доступних курсів, перегляд підписок, тестів, екзаменів, а також повідомлень.

Після переходу на сторінку з курсом користувач може лише переглянути основну інформацію. Для відкриття інших опцій необхідно підписатися.

Після оформлення підписки студент може ознайомитися з лекцією, а також пройти тест на засвоєння матеріалу.

На рисунку 18 зображено меню тестування.

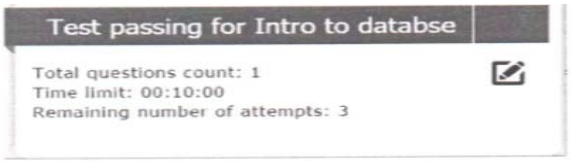

Рисунок 18 – Меню тестування

Після проходження тесту його результати можна переглянути перейшовши в «My Test» на дошці приладів. На рисунку 19 зображені результати проходження тесту.

Рисунок 19 – Результати проходження тесту

До інформації про тест входить назва лекції, дата та час проходження, результат у відсотках, а також статус перевірки.

ВИСНОВКИ

В результаті дослідження проведено аналіз проблеми розробки вимог та створення програмного забезпечення системи електронного навчання. Були розроблені функціональні та нефункціональні вимоги, на основі яких було створено веб-додаток на основі мікросервісів. Працездатність розробленого програмного забезпечення підтверджено за допомогою автоматичного (модульного та інтеграційного) і ручного тестування. Запропоноване програмно-архітектурне рішення може бути використане в системі дистанційного навчання.

ЛІТЕРАТУРА:

1. http://www.elitarium.ru/distancionnoe-obrazovanie-obuchenie-sistema-organizacija, 05.05.2018.
2. Afanasjev Yu.I. Sovremennie problem nauki i obrazovaniya, 2015 – S. 1–45.
3. Advanced Distributed Learning, Intelligent Tutoring System, and SCORM 2. – 2013. – P. 76–80.
4. Karl I. Vigers. Razrabotka trebovaniy k programmnomu obespecheniyu. – 2009 – S. 243–260.
5. Gang of Four Patterns //http://c2.com/cgi/wiki?GangofFour/, 20.04.2018.
6. Jess Chedvik. MVC ta rozrabotka realnih veb-prilozheniy s pomoshchju ASP.NET MVC. – M.: Wiljams, 2013 – 432 s.
7. Lucas Krause. Microservices: Patterns and Applications, - 2015.
8. Офіційна сторінка JetBrains // https://www.jetbrains.com/idea/features/, 01.05.2018.
9. Sinicin S.V, Nalutin N.Yu. Verifikacija programmnogo obespechenija. – M.: VINOM, 2008. – 368 s.
10. Lavrishcheva E.M. Metody I sredstva snzhenerii programmnogo obespecheniya: Uchebnik / E.M. Lavrishcheva, V.A. Petruhin. – M.: MFTS (GU), 2006. – 304 p.

Рецензент: д.т.н., проф. Коваленко В.Ф.,
Херсонський національний технічний університет